Constructive Solid Geometry with Projection: An Approach to Piano Movers' Problem

Natee Tongsiri

[1] Research Institute for Symbolic Computation
Johannes Kepler University Linz, Austria
natee.tongsiri@risc.uni-linz.ac.at *

[2] Department of Mathematics, Faculty of Science, Chiang Mai University
Chiang Mai, Thailand
natee@chiangmai.ac.th

Abstract. Configuration space obstacles are regions in a configuration space which represent forbidden configuration of the object due to the presence of other objects. They can be regarded as geometric objects which are representable using Boolean combinations of equations and inequalities. Moreover, they can also be represented using existential quantifiers which correspond to geometric projections. If the projected variables only occur algebraically then it is possible to eliminate quantifiers and represent configuration space obstacles in the semi-algebraic form. However, no matter how it is done, the quantifier elimination is computationally hard and the output in semi-algebraic representation is often large and cumbersome. Therefore, we are looking for a possibility to work directly with the quantified representation of the configuration space obstacles.

We investigate combination of tools which can be used in conjunction with this quantified representation. The main idea is the use of interval evaluation on equations and inequalities involving some transcendental functions. We suggested an extended version of Constructive Solid Geometry system which has trigonometric functions as well as the usual Boolean operators. The combination of this system together with interval arithmetic allows simplification and spatial subdivision and pruning to be done in a natural way. We also raise the possibility of an extended Constructive Solid Geometry system which would have projection and boundary formation as operators. This would allow compact representation of the configuration space, but presents computational problems which are, as yet, unsolved.

* Supported by the scholarship Technolgiestipendien Südostasien 2005/2006 (Postdoc) awarded by the Austrian Exchange Service – Agency for International Cooperation in Education and Research (ÖAD), Academic Cooperation and Mobility (ACM).

1 Background

Spatial planning problems are a class of geometric problems which involve placing an object among other objects or moving an object from one place to another without colliding with other objects in the process. We refer to the problem of placing an object among obstacles as a *Find-space* problem and refer to the problem of finding a collision-free path for an object as a *Find-path* problem or *Movers' problem*.

Movers' problem is a computationally difficult problem. Not only that the complexity of the computation increases with the number of dimensions and the number of objects involved, it also depends on the representation and the complexity of the objects. Much research has been devoted to the complexity aspect of the Movers' problem. For example, it was studied by Canny [8], Davenport [10], Hopcroft, Schwartz and Sharir [20], Lozano-Pérez [24], Reif [37, 38], Schwartz and Sharir [39, 40] and Vanderstappen, Halperin and Overmars [51].

Due to the high computational complexity of the problem, many applicable algorithms are probabilistic in nature. For example, Amato [1], Branicky [4, 5], Kavraki, Svestka, Latombe and Overmars [23]. However, there are no guarantee that these algorithms will find a collision-free path, even if one exists. This gave rise to a classification of levels of completeness for Movers' problem algorithms. For example, a *complete* algorithm correctly returns a path when one exists and declares that no path exists otherwise. An algorithm is *resolution complete* if it correctly returns a path when one exists at a chosen discretisation level and returns failure otherwise. An algorithm is *probabilistically complete* if a path exists but the probability that the method does not find one converges to zero as computation time increases.

1.1 Configuration Space Approach to Movers' Problem

In order to describe the position and orientation of the moveable object, one approach is to give this object a *reference point*. The position and orientation of this object in space can now be specified by a set of parameters which corresponds to the configuration of the object and its reference point, in relation to a reference frame, fixed in the environment. Thus a point, specified by these parameters, encapsulates both a position and an orientation of the original object. The space where the moveable object and the obstacles reside is often referred to as the *Workspace*. The parameter space of the moveable object is called *Configuration Space* or *C-space*.

This idea of mapping the original problem from lower dimension to a relatively simpler problem in higher dimension was first introduced by Udupa in [50] and later formalised by Lozano Pérez in [24]. By convention, a *position* of a translated object in the Workspace is the distance from the coordinate frame in each translational dimension to the reference point of the object. Similarly, the *orientation* of the object is specified by the angular displacement, relative to the original orientation of the object itself, in each rotational possibilities. Generally,

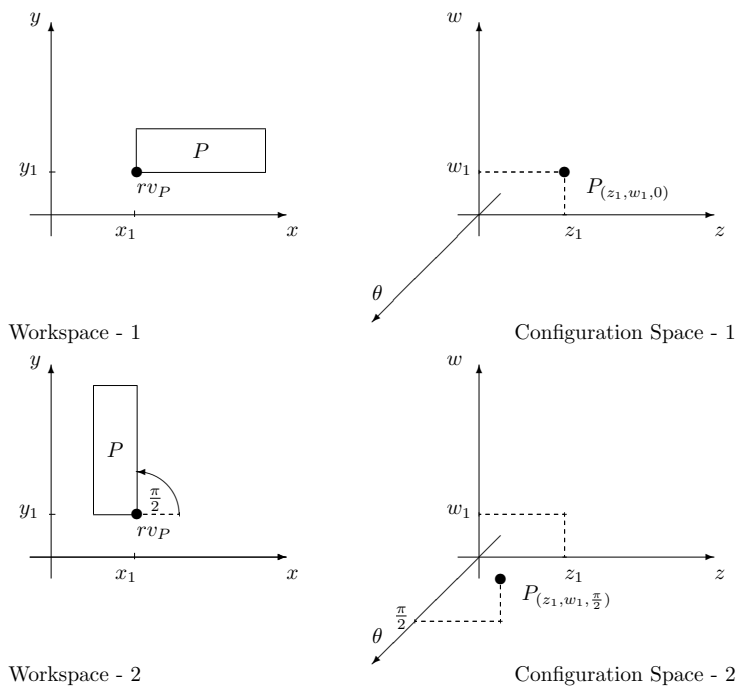


Fig. 1. A configuration of a polygon which may translate and rotate can be specified by three parameters; two parameters correspond to the two dimensions of the translation and one parameter corresponds to the rotation.

this orientation of an object is the anti-clockwise angle about the reference point of the object in each of the rotational dimension.

Figure 1 illustrates the relationship between Workspace and C-space of a 2-dimensional object P which can translate and rotate freely. The distance from the reference point rv_P relative to the origin of the coordinate frame and the degree of rotation around rv_P correspond directly to a position of a single point in C-space. Thus the configuration of P in the Workspace can be captured by three parameters (z, w, θ) where (z, w) correspond to the position of rv_P and θ corresponds to the angle of rotation of P about rv_P.

Dimension of C-space is governed by the dimension of the Workspace. In turn, the dimension of the Workspace is depended on the maximum degrees of freedom of an object. Since the degrees of freedom of an object is the number of 'ways' in which a particular object can 'move' in space, this is also the minimum number of independent parameters required to specify every conceivable position and orientation of an object, relative to a frame of reference. Since a configuration of a particular object refers to its position and its orientation, the degrees of freedom

of an object, which allow to translate and rotate, is a combination of degrees of freedom of its translational and rotational movement. An object moving freely in n-dimensional Workspace has at most n translational movements and $(\frac{1}{2}n(n-1))$ rotational possibilities. Therefore, it has at most $n + (\frac{1}{2}n(n-1))$ degrees of freedom.

Configuration Space Obstacles or *C-space obstacle* is a set of points in C-space which correspond to configurations which cause the moveable object to overlap with obstacles. Let P be a moveable object and let O_j be a set of static obstacles. All points in C-space that are not in C-space obstacle will correspond to the configuration of the objects P where it does not intersect with obstacles O_j. The collection of these points is called *C-free*. The boundary between C-space obstacle and C-free is referred to as the *contact surface*.

Definition 1. *Let C be a C-space. Define C-space obstacles of an object P due to an obstacle O as a collection of points $x \in C$ such that $P_{(x)} \cap O \neq \phi$, denoted by $CO_P(O)$.*

To solve a movers' problem, further work is needed. That is a path in the C-free needs to be constructed. The algorithm for Movers' problem using configuration space approach are often similar to Algorithm 1.

Algorithm 1 C-space Approach to Movers' Problem

Input: Bounding box B, Object P and Obstacle $O \in B$, P_{init}, $P_{goal} \in B$
Output: Continuous path from $P_{init} \in C$ to $P_{goal} \in C$ or $false$

1: Construct C-space C
2: Verify $P_{init} \in C$ and $P_{goal} \in C$
3: Establish C-space obstacle region in $CO_P(O) \in C$
4: Find continuous path $P_{path} \notin CO_P(O)$ from P_{init} to P_{goal}
5: **return** Path from P_{init} to P_{goal} or $false$

An important aspect of spatial planning systems that use C-space approach is the method of representing the C-space. The representation needs to be able to classify regions of C-space into at least 2 sub-regions; the C-space obstacle and C-free, and enable the search for positions within these regions. Many C-space representations exist, each has advantages and disadvantages over the other. Recent surveys, for example by Hwang [21] and Wise [52, 53], described many techniques used to compute and represent C-space which can be classified into many sub-categories, in many different ways. One of the major technique of representing C-space is cell decomposition which were studied extensively in, for example [7, 14, 24, 39–43]. Most technique is based on discretising the C-space into a finite number of cells and use some tests to classified each cells whether it belongs to the C-free or the C-space obstacle. By building a connectivity graph which represent adjacency relation of these cells, path planning become a graph-search problem in which many efficient algorithms exist.

1.2 Constructive Solid Geometry

Constructive Solid Geometry or CSG describes the geometry of a complex object by combining simple objects using operators of Set Theory. Primitive shapes can be expressed implicitly by polynomial inequalities. Object of zero thickness such as wires and sheets can be represented as well as solids. It also allows the use of *sine*, *cosine* and *exponential* functions when building descriptions of objects. Complicated CSG objects can be treated as though they were a single object and can be combined to make more complicated objects in the same manner. CSG method is useful both as a method for representing geometric object and as an intuitive user interface technique.

CSG is widely studied and CSG models are regarded as more stable than others because the properties of its Boolean operators are well understood. Gomes and Teixeira described in details in [16], the mathematical framework for computable CSG primitives using levels of decreasing abstraction from Boolean algebra of sets, set-point topology and geometry to semi-algebraic sets. The theory of Boolean algebra can be extended to the theory of sets. For example, a Boolean algebra $\langle S; \cap, \cup, \backslash \rangle$ in which elements of S are sets and the operations on sets are union ($\cup$), intersection ($\cap$) and complement($\backslash$), is called a Boolean algebra of sets. The concept of Boolean algebra of sets is useful for geometric modelling since the basic idea of CSG is also to represent sets in $\mathbb{R}^n$ by Boolean combinations of primitive sets. If the geometric objects which need to represented can be considered as sets then Boolean algebra of geometric objects allows us to define primitive geometric objects as sets and combining them together using Boolean operators.

A major approach of CSG to spatial planning is to construct C-space obstacles, which can be regarded as geometric objects in C-space, as a Boolean combination of CSG primitives. Each point inside this object represents the position of the actual object that causes collision with the obstacles in the Workspace. Given C-space obstacles, Find-space and Find-path correspond to the simpler problems of finding a single point, a position of the object, and a path, a sequence of position of the object, outside the obstacles.

Wise investigated in [53] the application of CSG to the problem of generating the global C-space map and demonstrated two approaches which semi-algebraic CSG can be used to compute global map of the C-space for a system of rigid bodies. The first is an approximate calculation of C-space obstacles and the second is a precise representation of C-space obstacles by formulating the contact surfaces analytically. The potential for combining the two approaches were also highlighted.

1.3 Interval Methods

One fundamental problem in computing is of representing real numbers accurately. In many cases, it is sufficient to compute with countable subsets of the reals, such as integers and rational numbers because computation is always possible as long as the requested memory is available. However, the computing time

of arithmetic operations of these subsets depends on the length of the arguments which are not always constant. Thus computing with such set of numbers has limited efficiency.

For a more efficient computation, accuracy has to be compromised. That is, real numbers have to be approximated and represented by a fixed-length representation. This finite subset of real numbers is often called *represented numbers*. Clearly, the input numbers and intermediate results of algorithms have to be approximated before they can be further computed. Additionally, arithmetic operations of represented numbers generally yield a represented number with some errors. These errors can accumulate during the chain of computations, making it a non-trivial problem to estimate the magnitude of the error in the final output.

Interval Arithmetic has been introduced by Moore [28] to control the round off errors of numerical computations and uncertainty in representing real numbers with floating points numbers. In the interval arithmetic framework, a real number is represented by two floating point numbers. These intervals are called *represented intervals*. Similar to the the case of computing with represented numbers, arithmetic operations on represented intervals generally yield represented intervals with some approximation. However, with some careful considerations of the operations, it is possible to guarantee that the final output of a chain of computation contains the correct real numbers.

Let $\mathbb{R}$ be the real line and let $\mathbb{F} \subset \mathbb{R}$ be a finite subset of rational number corresponding to rational or floating-point numbers in a certain binary representation. For every $a \in \mathbb{R}$ let $\overline{a}$ be the smallest element in $\mathbb{F}$ such that $\overline{a} \geq a$, and $\underline{a}$ be the greatest element in $\mathbb{F}$ such that $\underline{a} \leq a$. Called $\overline{a}$ the left bound and $\underline{a}$ the right bound. A finite interval on the real line is a subset of $\mathbb{R}$ defined in terms of end-points $\underline{a}$ and $\overline{a}$. Given $\underline{a} \in \mathbb{F}$ and $\overline{a} \in \mathbb{F}$ such that $\underline{a} \leq \overline{a}$, let the interval $[\underline{a}, \overline{a}]$ be a closed and connected subset of real numbers $\{x | x \in \mathbb{R} \ \wedge \ \underline{a} \leq x \leq \overline{a}\}$. An interval can be regarded as a finite region of one dimension. We refer to a region defined in this way as a *box*. Let I denote the set of intervals. The left bound of an interval I is denoted by $\underline{I}$ and the right bound by $\overline{I}$.

Definition 2. *For all* $[\underline{X}, \overline{X}], [\underline{Y}, \overline{Y}] \in \mathbb{IR}$ *the natural interval extensions of the elementary operations of arithmetic are*

$$
\begin{aligned}
[\underline{X}, \overline{X}] + [\underline{Y}, \overline{Y}] &= [\underline{X} + \underline{Y}, \overline{X} + \overline{Y}] \\
[\underline{X}, \overline{X}] - [\underline{Y}, \overline{Y}] &= [\underline{X} - \overline{Y}, \overline{X} - \underline{Y}] \\
[\underline{X}, \overline{X}] \times [\underline{Y}, \overline{Y}] &= [min(\underline{X}\underline{Y}, \underline{X}\overline{Y}, \overline{X}\underline{Y}, \overline{X}\overline{Y}), max(\underline{X}\underline{Y}, \underline{X}\overline{Y}, \overline{X}\underline{Y}, \overline{X}\overline{Y})] \\
[\underline{X}, \overline{X}]/[\underline{Y}, \overline{Y}] &= [min(\underline{X}/\underline{Y}, \underline{X}/\overline{Y}, \overline{X}/\underline{Y}, \overline{X}/\overline{Y}), max(\underline{X}/\underline{Y}, \underline{X}/\overline{Y}, \overline{X}/\underline{Y}, \overline{X}/\overline{Y})], \\
&\quad \textit{undefined if } 0 \in [\underline{Y}, \overline{Y}].
\end{aligned}
$$

A box can be defined as a tuple of interval $I = (I_1, \ldots, I_k)$ which is the Cartesian product $I_1 \times I_k$. For example, two intervals, one along each Cartesian coordinate axes represent 2-dimensional coordinate aligned box, which is an

area between 4 line segments. In the same fashion, three interval can represent a cuboid. Additionally, given an interval I, let $w(I)$ be the width of I defined by $\overline{(\overline{I} - \underline{I})}$. By convention, the width of a box is the width of its biggest interval.

Denote the set of intervals over $\mathbb{R}$ by $\mathbb{IR}$ and the set of n dimensional interval vectors by $\mathbb{IR}$. Every continuous function $f : \mathbb{R}^n \to \mathbb{R}$ can be extended to $\mathbb{IR}^n \to \mathbb{IR}$ by

$$f(I) = \{f(x)|x \in I\}.$$

This allows us to embed $\mathbb{R}$ into $\mathbb{IR}$ by the morphism $m(x) = [x, x]$ which is also defined for each component for tuples of intervals.

1.4 Movers' Problem and Quantifier Elimination

Quantifier elimination in the first order theory of real closed fields is a decision procedure of solvability of constraints over real numbers. It can also be used for eliminating non-query variable from answer constraints and to process the solution set in order to provide convenient solutions.

Let $L_{\mathbb{R}}$ be the first order language of the ordered field of the reals. A *formula* of this language is the expression which are built up from atomic formulae using the logical operators $\wedge, \vee, \neg$. An *atomic formula* is the expression of the form $p(x_1, \ldots, x_n) \ \diamond \ 0$ where $p(x_1, \ldots, x_n)$ is a polynomial in variables $x_1, \ldots, x_n$ with integral coefficients and $\diamond \in \{>, \geq, \neq, <, \leq\}$. Additionally, some or all of the variables in a formula may be quantified over the field by universal ($\forall$) and existential ($\exists$) quantifier.

An occurrence of a variable x in A is *bound* if it occurs in a sub-formula B of the form $\exists x[B(x)]$ or $\forall x[B(x)]$. An occurrence of a variable is *free* if it is not bound. In the above example, occurrences of variable x_1 and x_2 are bound and the occurrence of the variable x_3 is free. Suppose the formula $A(x_1, \ldots, x_n)$ has free variables among $x_1, \ldots, x_n$. This formula is semi-algebraic and defines a set $\{(x_1, \ldots, x_n) : A(x_1, \ldots, x_n)\} \subset \mathbb{R}^n$.

A formula in which all variables are quantified is called a *sentence* which has a definite truth value. When free variables are substituted by specific values leaving only bound variable in the formula, it becomes a sentence. A set of values is a *solution* for the formula if the sentence, obtained by substituting all variables in the formula by the values, is true. Two formulae are *equivalent* if they have the same solutions.

In 1930, Tarski showed in [47] that all quantified formulae can also be defined without quantifiers and presented an algorithmic quantifier elimination (QE) method. The algorithm accepts any formula of real closed field as an input and outputs an equivalent formula containing the same variables with no quantifiers. The output formula is true for the same values of its free variables as the input formula. If the input formula contains free variables then the output formula expresses a necessary and sufficient algebraic condition of the input formula to hold. For example, applying the quantifier elimination to the formula of L_R: $a \neq 0 \wedge \exists x[ax^2 + bx + c = 0]$ produces the well-known necessary and sufficient condition $a \neq 0 \wedge b^2 - 4ac \geq 0$ where a, b, c are free variables.

In 1973, Collins [9] has used Cylindrical Algebraic Decomposition (CAD) to eliminate quantifiers. The CAD method for quantifier elimination consists of three main phases. The first phase extracts the polynomials occurring in the input formula and factoring them into irreducible factors, assuming that each atomic formula of the input formula is of the form $p = 0$ or $p < 0$ where p is a multivariate polynomial with integer coefficients. The second phase constructs a decomposition of real r-dimensional space, where r is the number of variables in the formula, into a finite number of connected regions, called *cells*. Each polynomial in all cells is invariant in sign. These cells are arranged in a certain cylindrical manner. From this cylindrical decomposition, it is then quite straightforward to apply the quantifiers by using a sample point from each cell to determine the invariant truth value of the input formula in that cell. This application of quantifiers reveals which cells in the subspace of the free variables are true. The final phase constructs an equivalent quantifier-free formula from this knowledge.

In Collins' original method, this problem was solved by a method called augmented projection that provided a quantifier-free formula for each of the true cells. In 1990, Hong [18] has devised a generally more efficient method which appears to work in most cases. In 1995, Hong [19] also proposed and implemented a more efficient approximate quantifier elimination using interval arithmetic. The algorithm was later improved and implemented by Ratschan [34–36]. However, beyond linear case and a few variables, quantifier elimination is very difficult, or produces such large output as to be hard to manage. Additionally, it is only possible in principle to eliminate quantifier if the bound variables only occur algebraically.

The mathematical and computational structures of the spatial planning problem when stated in algebraic terms are reasonably well-understood [8]. Objects defined by semi-algebraic sets are highly flexible and expressive but they are computationally expensive to compute. Existential quantifiers, which correspond to geometric projection, and semi-algebraic sets with Boolean operators can be used to describe C-space obstacles. We can outline the method as follow:

1. Represent C-space obstacle in quantified form.
2. Decompose C-space into semi-algebraic cells so that each cell is either entirely contained in the C-space obstacle or disjoint from it.
3. Determine adjacency relation between cells in C-space obstacle.
4. Represent obstacle connectivity as finite graph.
5. Solve Find-path problem by standard graph searching techniques.

Both step 2 and 3 are difficult in practice. For example, in step 3, the set of collision-free points is a semi-algebraic set that can be determined by quantifier eliminations. Since it is necessary to decide all cells adjacencies in the CAD in order to determine whether two points are in the same component, one approach is to decide if cell C_1 is adjacent to cell C_2. Cell C_1 and C_2 are adjacent if $closure(C_1)$ intersects C_2 or if C_1 intersects $closure(C_2)$. The closures can be expressed in quantified form, and then the quantifiers can be eliminated.

Several methods based on generating a cylindrical cell decomposition of C-free were first proposed by Schwartz and Sharir [40]. Attempts to solve simple two dimensional spatial planning problem using CAD to eliminate quantifiers, for example, by Davenport [10], Davenport, Siret and Tournier [11], Kalkbrener and Stifter [22], McCallum [27], Sturm and Weispfenning [46], achieved limited results because of excessive computational resource requirement.

1.5 Binary Space Partitioning

Binary Space Partitions tree or BSP tree which was introduced by Fuch, Kedem and Naylor in [15] offers a simple way to implement a geometric divide-and-conquer strategy. The method of BSP divides the space into two parts, with a hyperplane or subplane. This action may bisect the objects if necessary. Parts of the objects belongs only to one of side of the bisection. The two resulting parts may get divided recursively in a similar manner. The process continues until some conditions are met. This division process can be naturally represented as a binary tree. A node represents a part of the space and stores the cut splits the space into two parts and both children represent the spaces. Consequently, the number of regions in the final configuration is the number of leaves in the BSP tree. Each leaf of the BSP tree represents the final partitioning of the space and usually stores very small fragments of the original object. Such rectangle leaves form a very important class of objects in application domains because complex objects can often be replaced by their bounding rectangles.

Since the number of leaves in the trees is a fundamental aspect for most BSP applications, the problem of bounding the size of BSP trees is an important research topic. There has been a considerable interest in proving tight lower and upper bounds on the exact size of the BSP trees for various objects. For example in Berman, DasGupta and Muthukrishnan [2], Dumitrescu, Mitchell, and Sharir [13], Paterson and Yao [32, 33]. In CSG related application, Thibault and Naylor demonstated in [48] that regular sets can be represented using BSP trees and how BSP trees can also provide an exact representation of arbitrary polyhedra of any dimension.

2 Geometric Models of Configuration Space Obstacles

Configuration space obstacles of the moveable object among obstacles can be regarded as geometric objects. The solid part of the geometric model could correspond to the configuration that would cause the object to collide with the obstacles where empty space outside the solid corresponds to the possible configuration of the object in the Workspace.

The idea is to characterise the position and the geometric constraints of the object due to the obstacles by some conditions imposed on the entire object by the presence of the obstacle in the same Workspace. This can be done by stating the conditions of every point of the object using quantifiers.

For example, to represent two dimensional moving object P and obstacles O in two dimensional Workspace using implicit representation can be proceed as followed.

For a point $z, w \in \mathbb{R}^2$, define P and O as subsets of the Workspace $\mathbb{R}^2$ by $P = \{(z, w) : p(z, w)\}$ and $O = \{(z, w) : o(z, w)\}$ where $p(z, w)$ and $o(z, w)$ are some conditions for (z, w). That is, in the Workspace the moving object is a set of points $\{(z, w) \mid P(z, w)\}$ and the obstacle is $\{(z, w) \mid O(z, w)\}$.

It is clear that a position of the object P is impossible if and only if, there is a point (z, w) in P such that, if this point is translated by (x, y) and rotated about the reference point rv_P by θ, it will be in O. Such a point represents the overlapping of the moving object and the obstacle. The collection of these points forms a C-space obstacle.

The above statement can be represented in quantified form as:

$$\exists z \exists w (P(z, w) \ \wedge \ O(x + (zcos\theta - wsin\theta), y + (zsin\theta + wcos\theta)) \ )$$

Thus the C-space obstacle is a set of points

$$\{(x, y, \theta) \mid \exists z \exists w (P(z, w) \ \wedge \ O(x + (zcos\theta - wsin\theta), y + (zsin\theta + wcos\theta))\}$$

To represent C-space obstacles in this way leads to an object that has more variables than the degrees of freedom of the problem. The Boolean combinations of the formulae with variables z, w, x, y, θ define a geometric object called the *omnimodel*. The space which contain the omnimodel is referred to as the *omnispace* [29–31].

Assume now that $O(z, w)$ and $P(z, w)$ do not involve any of the trigonometric or exponential functions. By using quantifier elimination on the above representations, an extended semi-algebraic representation of the C-space obstacle in (x, y, θ) can be obtained. The C-space obstacle can be represented as the result of a projection from 5-dimensional space with coordinates (z, w, x, y, θ) to the 3-dimensional space with coordinates (x, y, θ). The variables z and w were projected out.

However, quantifier elimination is computationally hard, and the output in semi-algebraic representation is large and cumbersome. It seems that we should learn to work directly with the implicit representation of the C-space obstacle. In any case, if the moving object or part of the obstacle is not algebraic, we must be able to deal with quantified representation of the C-space obstacle, since elimination of quantifiers may not be possible.

2.1 Alternative Representation

By convention, P is usually connected, that is P is always either entirely outside the obstacle or entirely inside the obstacle or is intersected with at least an edge of the obstacle. Thus the position of the object P is also impossible if and only if, either there is a point (z, w) in P such that, if this point translated and

rotated, it will be on the edge of the obstacle, or the reference point of P is in the obstacle. We can also represent the collection of these points as:

$$O(x, y, \theta) \vee \exists z \exists w (P(z, w) \wedge edge(O(x + (zcos\theta - wsin\theta), y + (zsin\theta + wcos\theta))))$$

Here $edge(z, w)$ refers to extended semi-algebraic representation of the edges of the obstacle. We call this representation *edge formulation*. This *edge* operator is the boundary formation operator which need to be defined.

3 Extended Semi-algebraic Sets as CSG Primitives

CSG models in higher dimensions can represent configuration-space obstacles. Simple objects in CSG are referred to as *primitives*. In order to represent CSG primitives, we need to consider a computable representation. Any set defined by multivariate polynomial inequality or equality involving some transcendental functions appears to be an appropriate candidate since they have good expressive power. For example, primitives such as half-spaces, spheres and cylinders are easy to represent in semi-algebraic form. Additionally, polynomial functions together with sine and cosine is a natural tool to describe geometric constraints.

Semi-algebraic sets are subsets of some $\mathbb{R}^n$ defined by Boolean combinations of a finite number of polynomial equations and inequalities. In 1930, Tarski [47] showed that every set in some $\mathbb{R}^n$, which can be defined using Boolean operators and quantifiers starting with polynomial equalities and inequalities, is semi-algebraic. Existential quantifiers correspond to geometric projection. Thus semi-algebraic sets are closed under finite union, intersection, negation and projection. Geometric objects can be built not only from Boolean combinations of equations and inequalities but also with projection, i.e. using existential and universal quantification. This makes it possible to represent objects and their motion constraints.

Let $Z[x_1, \ldots, x_n]$ denotes the set of polynomials in variables $x_1, \ldots, x_n$ with integral coefficients. A *semi-algebraic primitive* is a subset of $\mathbb{R}^n$ which admits some representation of the form $\{(x_1, \ldots, x_n) : p(x_1, \ldots, x_n) \ \diamond \ 0\}$ where $p(x_1, \ldots, x_n) \in Z[x_1, \ldots, x_n]$ and $\diamond \in \{ \leq, <, =, \neq, >, \geq \}$. An expression of the form $p(x_1, \ldots, x_n) \ \diamond \ 0$ is called an *atomic formula*.

A *semi-algebraic set* is a semi-algebraic primitive or a Boolean combination of semi-algebraic primitives. The semi-algebraic sets of $\mathbb{R}^n$ form a Boolean algebra with $+$ as set union ($\cup$), $\cdot$ as set intersection ($\cap$), $-$ as a set complement ($-$), 0 as an empty set (ϕ) and 1 as the universal set ($\mathbb{R}^n$).

Boolean algebra of semi-algebraic sets provides a finite description of geometric objects and a set of operators capable of manipulating them. Semi-algebraic sets can be used as a CSG primitive. Object in CSG can be viewed as a set-theoretic composition of elementary semi-algebraic sets in $\mathbb{R}^n$. By definition, the Boolean combinations of semi-algebraic sets are closed under elementary set-theoretic operators; finite intersection, finite union and complement. We can use these semi-algebraic sets and its operators to define geometric objects.

Since we would like to allow, as a CSG primitive, any set defined by multivariate polynomial inequality or equality involving some transcendental functions such as sine and cosine, we have to extend the CSG primitives beyond the semi-algebraic.

Let

$$Z[x_1, \ldots, x_n, \sin(x_1), \ldots, \sin(x_n), \cos(x_1), \ldots, \cos(x_n)]$$

denote the set of polynomials with integral coefficients in $x_1, \ldots, x_n$, and the sines and cosines of these variables. An *extended semi-algebraic primitive* is a subset of $\mathbb{R}^n$ which admits some representation of the form

$$\{(x_1, \ldots, x_n) : p(x_1, \ldots, x_n) \ \diamond \ 0\}$$

where $p(x_1, \ldots, x_n) \in Z[x_1, \ldots, x_n, \sin(x_1), \ldots, \sin(x_n), \cos(x_1), \ldots, \cos(x_n)]$, and $\diamond \in \{ \leq, <, =, \neq, >, \geq \}$.

An *extended semi-algebraic set* is an extended semi-algebraic primitive or a Boolean combination of extended semi-algebraic primitives. Boolean algebra of extended semi-algebraic sets, which includes trigonometric functions, has good expressive power. Not only that it is capable of representing static objects but it is also natural to describe motion constraints of objects in this form. However, the subsets of $\mathbb{R}^n$ which can be represented by Boolean combinations of the extended primitives do not always have good closure properties. For example, although we can represent the trigonometric functions, we cannot, as far as we know, represent the primitive $\{(x, y) : y - \sin(x^2) \leq 0\}$.

4 CSG System and Interval Methods

It is clear that, Boolean algebra of extended semi-algebraic sets provide a finite description of geometric objects and a set of operators capable of manipulating them. Call this system *CSG system*.

The configuration space obstacles are geometric objects that can be represented using Boolean combinations of inequalities. Moreover, they can also be represented using existential quantifiers which correspond to geometric projections. If the projected variables only occur algebraically then it is possible to eliminate quantifiers and represent configuration space obstacles in the semi-algebraic form. However, no matter how it is done, the quantifier elimination is computationally hard and the output in semi-algebraic representation is often large and cumbersome. Therefore we are looking for a possibility to work directly with the quantified representation of the configuration space obstacles using interval arithmetic.

Since its inception, interval computations were employed in many different manners and in many different application fields. We are interested in using interval mainly as a tool to approximate the value of some functions over regions of a space. There are several investigations which employed interval analysis in this fashion. For example Bowyer, Berchtold, Eisenthal, Voiculescu, and Wise [3], Duff [12], Martin, Shou, Voiculescu, Bowyer and Wang [26], Snyder [44] and

Tupper [49] discussed how interval analysis could be a useful tools for geometric modelling and can be used to solve a wide variety of problems in computer graphics, including ray tracing, plotting algebraic curve, graphing two-dimensional implicit equations and inequalities, interference detection, polygonal decomposition of parametric surfaces, and Constructive Solid Geometry on solids bounded by parametric surfaces.

The property of C-space obstacle in which:

$$CO_P(O_1 \cup O_2) \equiv CO_P(O_1) \cup CO_P(O_2)$$

where P is the object and O_i are obstacles, gives rise to the idea of generating C-space obstacle of many simple objects and combining the results together. Branicky showed in [6] that the validity of this property holds regardless of convexity or connectedness of the actual obstacles. This property is particularly useful with the CSG representation of the C-space obstacles since it is natural to build complicated object using Boolean combinations of many simple ones.

Complicated CSG objects can be represented by a tree structure with Boolean operators on the internal nodes, and primitives at the leaves. The usual operators include $\cup$ (or), $\cap$ (and), $\neg$ (not), and $-$ (difference). To answer a query about an object represented in this way is computationally expensive since every node and leaf has to be consulted. One way to overcome this problem is based on the assumption that 'representation of objects may be globally complicated but locally simple', objects can be much simpler within "small enough" region. The restrictions of the number of nodes and leaves that need to be considered can be imposed by dividing the space that contains the tree into smaller spaces, and by pruning the tree for each space. Pruning and subdivisions can lead to simpler representations of objects. Each time the subdivision occurs, the smaller sub-space may have simpler objects while the combinations of all spaces still represent the more complicated original model. Although it is likely, there is no guarantee that the object will be simpler in a smaller region of interest. However, at least the union of all regions still represent the more complicated original model.

The method we used is a combination of two processes. The first divides the original region of interest into many regions; each has the original object inside. This process is called *spatial subdivision*. The second process reduces the number of primitives which made up the object in each region by systematically removing unnecessary primitives. This process is called *pruning*. The subdivision process can be applied recursively or adaptively, each time with the pruning process to simplify the object to its region, until a certain condition is met.

4.1 Models and Boxes

In order to use the pruning and recursive subdivision technique, we introduce the concept of models and boxes.

Definition 3. *A CSG model M is a tree structure with Boolean operations on the internal nodes and atomic formulae on the leaves.*

Additionally, we will assume some list $(x_1, \ldots, x_n)$ of variables which may appear in M. CSG object trees define subset of $\mathbb{R}^n$ but a CSG model trees such as M describe conditions of variable $(x_1, \ldots, x_n)$ available to them. For example, for a variable list (x_1, x_2, x_3) a model M could take the form similar to Figure 2.

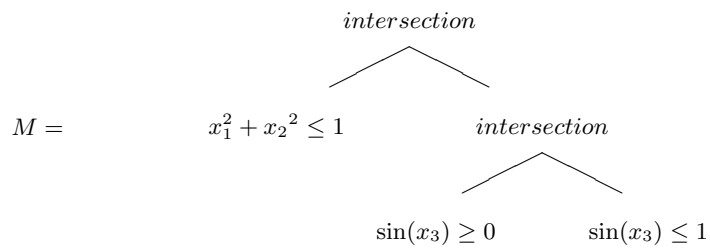


Fig. 2. Simple model tree.

The atomic formulae may define semi-algebraic primitives, or extended semi algebraic primitives. We might also wish to restrict the primitives to a subset of the semi-algebraic, such as, for example, linear half-spaces and cones. The geometric operations are n-ary union and intersection and unary complement.

Definition 4. *Let B be a list of m closed intervals $([a_1, b_1], \ldots, [a_m, b_m])$. Call B a box. Each interval in B represents a coordinate-aligned edge of the box. The correspondence is determined by the ordering of the intervals and the ordering of variables in a specified ordered list.*

Suppose an m-dimensional box is $([a_1, b_1], \ldots, [a_m, b_m])$ and the ordered list of variables are $x_1, \ldots, x_n$ where $m \leq n$. By convention, the interval $[a_1, b_1]$ corresponds to the variable x_1, the interval $[a_2, b_2]$ corresponds to x_2, and the correspondence carry on respectively to $[a_m, b_m]$ which corresponds to x_m. The variables $x_{m+1}, \ldots, x_n$, if exist will be ignored. Additionally, the ordered list of variables also label the coordinate axes of this set in $\mathbb{R}^n$ and if the length of the list of interval which defines the box is m, the box is said to be m-dimensional.

For example, suppose the ordered list of variables is (x_1, x_2, x_3, x_4) and the box B is $([a_1, b_1], [a_2, b_2])$. Figure 3 depicted B, a 2-dimensional coordinate-aligned box.

The box can also be represented in semi-algebraic form. For example, suppose the ordered list of variables is $x_1, \ldots, x_n$ and a box B of closed intervals is $([a_1, b_1], \ldots, [a_m, b_m])$ where $m \leq n$. The box B can be written in semi-algebraic form as:

$$(x_1 \geq a_1 \wedge x_1 \leq b_1) \wedge \ldots \wedge (x_m \geq a_m \wedge x_m \leq b_m).$$

Definition 5. *Let M be a model with variable from $x_1, \ldots, x_n$ and let B be an m-dimensional box where $m \leq n$. Define the model M over the box B to be the set of points in B that satisfy M. Denote this set in $\mathbb{R}^n$ by (M, B).*

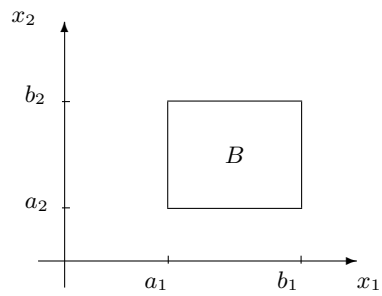


Fig. 3. A coordinate-aligned box which corresponds to a list of two intervals.

Since the set (M, B) is the collection of points in B that satisfy M, $(true, B)$ is the box B itself and $(false, B)$ is the empty set. We will use a pair (M, B), where M is a model and B is a box, to define a subset of $\mathbb{R}^n$. This (M, B) is our primitive CSG object; the building block of our geometric language which can also be regarded as an extended semi-algebraic set.

It can happen that (M, B) defines a set which does not depend on one or more of the variables from the variable list. This is necessary if we want intersection to be defined in a natural way. We have described a model as a tree. If we climb up the tree, the number of variables visible below us may change. We do not in general wish to change space every time this happens. Therefore, we work always in subsets of the box B.

The sets defined by (M, B) form a Boolean algebra of the subsets of B. This set definition is recursive on the structure of M. That is, if M is an atomic formula, (M, B) will be the subset of B in which M is $true$ and it follows that:

$$(union(M_1, M_2), B) = (M_1, B) \cup (M_2, B)$$
$$(intersection(M_1, M_2), B) = (M_1, B) \cap (M_2, B)$$
$$(complement(M), B) = B - (M, B)$$

Note that we distinguished between the two types of tree structure by the introduction of the following notations:

	CSG Tree	Model Tree
Primitives	Semi-algebraic Sets	Atomic Formulae
Operators	$\cup$	*union*
	$\cap$	*intersection*
	$-$	*complement*

4.2 Evaluation

For each pair (M, B), the box B limits the scope in which the model M is defined. It also provides intervals which correspond to variables in M from which we can

determine the value of each atomic formula in the model tree by using interval arithmetic.

In order to reduce the number of atomic formulae of a model in a box, all the formulae that make up the model need to be evaluated. This is to determine the value of each atomic formula over the box and decide if it can be simplified. The evaluation is done by using interval arithmetic on each atomic formula. By substituting each variable of the formulae with the corresponding interval, the range of values of the function in the atomic formula can be calculated. This range will also be an interval. The value of the atomic formula can be evaluated according to its operator. This process is referred to as *evaluating the formula over a box*. The value of the atomic formula after the substitution is either $true$, $false$ or *undecided*.

Table 1. Value of atomic formula according to the range of intervals

	$[-,0]$	$[0,+]$	$[-,+]$	$[-,-]$	$[+,+]$
$p=0$	*undecided*	*undecided*	*undecided*	$false$	$false$
$p<0$	*undecided*	$false$	*undecided*	$true$	$false$
$p\leq 0$	$true$	*undecided*	*undecided*	$true$	$false$

Table 1 shows the value of output intervals corresponding to the operator of the formula. The simplifications of models occur when some formula evaluate to $true$ or $false$ and there is no simplification when the formula evaluated to *undecided*.

Since this set definition is recursive on the structure of M, when M is an atomic formula, (M, B) will be the subset of B in which M is $true$. That is, when the formula evaluated to $true$ it can be interpreted as M is $true$ inside the box B and the intersection of the model and the box is the box itself. In this case we replace the formula with $true$. When the formula evaluated to $false$, M is $false$ inside the box B and we can replace it with $false$. The original primitive is returned if the evaluation result is *undecided*.

For example, consider $M = 2x-3y+4z+12 \leq 0$ and the box $([-1,4],[1,3],[0,4])$. Over the box the M, in this case – an atomic formula, is:

$$\begin{aligned} 2([-1,4]) - 3([1,3]) + 4([0,4]) + 12 &\leq 0 \\ [-2,8] - [-9,-3] + [0,16] + [12,12] &\leq 0 \\ [1,33] &\leq 0 \end{aligned}$$

which evaluated to $false$.

This process can be thought of as a function which takes an atomic formula P and a box B as the arguments and return either a value $true$, $false$ or the original primitive along with the box:

$$Eval(P,B) \rightarrow \{(true,B),(false,B),(P,B)\}.$$

By convention, an atomic formula of a model M will be evaluated to *undecided* if the atomic formula has an instance of a variable which does not correspond to an interval.

4.3 Pruning

The evaluation process maybe able to replace some formulae of the model in the box by $true$ or $false$. As a consequence, this can lead to a reduction of the number of formulae that made up a model. This is because a model consisted of unions, intersections and complements of formulae. Once each formula in the model has been evaluated and some replaced with $true$ or $false$, we can work up the model tree applying the operators to each leaf and achieve some simplification.

The pruning process ensures that inside the box there are no unnecessary formulae while the simplified model is still representing the original one. We can simplify the model over the box by applying the rules of the operators to the primitives.

The simplification rules of the CSG Boolean operators are:

$$\begin{aligned} union(undecided, true) &= true \\ union(undecided, false) &= undecided \\ intersection(undecided, true) &= undecided \\ intersection(undecided, false) &= false \\ complement(true) &= false \\ complement(false) &= true \\ complement(undecided) &= undecided \end{aligned}$$

The process of pruning the model to the box takes a model and a box (M, B) and produces another model over the same box (M', B). In the box B, the new model M' is either the same as M or simpler than M. Also, (M, B) and (M', B) define the same set.

Algorithm 2 is an example of the pruning procedure. It starts at the root of the model and recursively works down to all the leaves. While the node is still an operator, the evaluation is deferred by calling the procedure $Prune()$ again.

Since CSG operators are n-ary union, n-ary intersection and unary complement, the algorithm can exploit many known properties. For example, it can take into account the simplification that can be made in the case where one of the operand of *union* simplified to $true$ and similarly, where one of the operand of *intersection* simplified to $false$. If the node is not one of CSG operators then we have reached the leaf and procedure $Eval()$ is called to evaluate that particular formula to the box.

4.4 Recursive Subdivision

Since we are representing boxes using intervals, we consider using isothetic or axially-aligned box division for it is straightforward to perform subdivision tech-

Algorithm 2 $Prune(M, B)$

Input: (M, B)
Output: (M', B) $\{(M', B)$ defines the same set as $(M, B)\}$

```
S ← φ
if (M = union{M_1, ..., M_k}) then
    for i = 1 to k do
        (M'_i, B) ← Prune(M_i, B)
        if (M'_i = true) then
            return(true, B)
        else if (M'_i ≠ false) then
            S ← union(S, M'_i)
        end if
    end for
    if (S = φ) then
        return(false, B)
    else
        return(S, B)
    end if
else if (M = intersection{M_1, ..., M_k}) then
    for i = 1 to k do
        (M'_i, B) ← Prune(M_i, B)
        if (M'_i = false) then
            return(false, B)
        else if (M'_i ≠ true) then
            S ← intersection(S, M'_i)
        end if
    end for
    if (S = φ) then
        return(true, B)
    else
        return(S, B)
    end if
else if (M = complement{M_1}) then
    (M'_1, B) ← Prune(M_1, B)
    if (M'_1 = true) then
        return(false, B)
    else if (M'_1 = false) then
        return(true, B)
    else
        return(complement{M'_1}, B))
    end if
else
    return(Eval(M, B), B)
end if
```

nique and to determine adjacencies between boxes. This spatial subdivision process of the original regions of interest takes the form called Binary Spatial Partition (BSP). The BSP tree is a binary tree representing a recursive partitioning of n-dimensional space hyperplane or sub-plane, for any dimension n. Regions which are n-dimensional coordinate-aligned boxes defined by n closed intervals are divided equally into two adjacent regions along an axis. These intervals are subsets of $\mathbb{R}$ defined in terms of their end-points. A closed interval $\{x|x \in \mathbb{R}\ and\ a \leq x \leq b\}$ usually denoted by $[a, b]$.

We first consider a simple subdivision technique called recursive subdivision which divides a box into two sub-boxes and the divisions carry on recursively until the termination conditions are met. There are two subdivision decision strategies; *blind* and *adaptive*. The blind strategy determines the position of the subdivision by relatively fixing the point for each sub-division, for example, subdividing at a mid-point between two vertices. The adaptive strategy determines subdivision points by taking the contents of the box into account.

Regardless of the division strategy, when a box is divided, we get two or more adjacent sub-boxes with the same model inside. Each model can then be simplified over its own box by the pruning process and hopefully will be simpler than the original model. In this way, at any point of the process, the union of all sub-boxes with their models results in the original box with the original model.

Since the set definition of (M, B) is recursive on the structure of M, if $B = union(B_1, B_2)$ it follows that: $(M, B) = union((M, B_1), (M, B_2))$

For example, given a box B and a model $M = union(intersection(M_1, M_2), M_3)$, if the box is subdivided into two sub-boxes, B_1 and B_2 along one of its coordinate, then:

$$\begin{aligned}(M, B) &= (union(intersection(M_1, M_2), M_3), B) \\ &= union(\\ &\quad union(intersection((M_1, B_1), (M_2, B_1)), (M_3, B_1)), \\ &\quad union(intersection((M_1, B_2), (M_2, B_2)), (M_3, B_2)))\end{aligned}$$

The algorithm, which divides the longest side of the box, can be expressed in algorithmic form as in Algorithm 3.

After the pruning process, the union of the sets defined by (M, B_1) and (M, B_2) is the same as the set defined by (M, B). The division can be carried on recursively along with the pruning process until some conditions are met. There are a few options to consider as a termination condition of the process. For example, the recursive subdivision could stop when all the boxes are sufficiently small. It is also possible to determine if the model is simple enough that is there are a certain number of atomic formulae left in the model.

The Algorithm 4 recursively subdivides the longest side of the box until the box is small enough or the model is simple enough.

When we extend the primitive to extended semi-algebraic sets, pruning and recursive subdivision are still applicable. This is because the evaluation is done by interval arithmetic which *sine*, *cosine* and *exp* functions are also defined.

Algorithm 3 $SubDivide(M, B)$

Input: M, B
Output: $(M, B_1), (M, B_2)$

$position \leftarrow MaxSidePosition(B)$
$interval \leftarrow Part(B, position)$
$lower \leftarrow LowerEnd(interval)$
$upper \leftarrow UpperEnd(interval)$
$midpoint \leftarrow lower + Size(interval)/2$
$B_1 \leftarrow Part(B, position) \leftarrow [lower, midpoint]$
$B_2 \leftarrow Part(B, position) \leftarrow [midpoint, upper]$
$return\ (M, B_1), (M, B_2)$

Algorithm 4 $RecurSubDivision(M, B)$

Input: M, B
Output: $(M_1, B_1), \ldots, (M_k, B_k)$

$(M', B) \leftarrow Prune(M, B)$
if $IsSimple(M')$ *or* $IsSmall(B)$ **then**
  $return(M', B)$
else
  $(M, B_1), (M, B_2) \leftarrow SubDivide(M, B)$
  $return(RecurSubDivision(M, B_1), RecurSubDivision(M, B_1))$
end if

5 Extended Operators

Whether or not we extend the semi-algebraic primitives to include trigonometric functions, we may wish to extend the set of operators. We now wish to add two new operators: boundary and projection, to represent complex objects using extended semi-algebraic primitives.

5.1 Boundary Operator

Definition 6. *Let S be a subset of $\mathbb{R}^n$. Define $Closure(S)$ to be the set of points which are either in S or which have elements of S arbitrarily near them.*

For example, if $S = \{x : x < 0\}$ then $Closure(S) = \{x : x \leq 0\}$ whereas if $S = \{x : x = 1\}$ then $Closure(S) = \phi$.

Definition 7. *Let S be a subset of $\mathbb{R}^n$. Define*

$$Boundary(S) = Closure(S) \cap Closure(\overline{S})$$

where $\overline{S} = \mathbb{R}^n - S$. A point is in $Boundary(S)$ if it is arbitrarily close to points in S and also close to points in $\overline{S}$.

However, since set complement operator depends on the universe, so does the boundary operator. If the universe is B and $S \subseteq B$ then

$$Boundary(S) = Closure(S) \cap Closure(B - S).$$

Additionally, if S is a semi-algebraic set then $Closure(S)$ and $Boundary(S)$ are semi-algebraic.

For example, for $S_1, S_2 \subseteq \mathbb{R}^2$ the set $S_1 = \{(x, y) : x^2 + y^2 - 1 \leq 0\}$ defines a disc and $Boundary(S_1) = \{(x, y) : x^2 + y^2 - 1 = 0\}$ defines the circle in $\mathbb{R}^2$. Similarly, the set $S_2 = \{(x, y) : y \leq x^2\}$ defines a region below the parabola and $Boundary(S_2) = \{(x, y) : y = x^2\}$ defines the parabola in $\mathbb{R}^2$.

5.2 Projection Operator

Define the projection operator as follows:

Definition 8. *Let S be a subset of $\mathbb{R}^n$. Suppose $V = \{y_1, \ldots, y_k\}$ is a subset of the variables $\{x_1, \ldots, x_n\}$. Let $W = \{w_1, \ldots, w_j\}$ be the variables in $\{x_1, \ldots, x_n\}$ but not in V. Define a projection of variables in V of set S as:*

$$Projection_V(S) = \{(w_1, \ldots, w_j) : (x_1, \ldots, x_n) \in S\} for\ some \{y_1, \ldots, y_k\}.$$

Define cylindrical projection of variable in V of set S as:

$$CylProjection_V(S) = \{(x_1, \ldots, x_n) : (w_1, \ldots, w_j) \in Projection_V(S)\}$$

Both projection operators have two arguments, a subset of $\mathbb{R}^n$ and a set of variables to be projected out. The interpretation of $Projection$ is as the usual projection operator where the result of the projection is in the space of the remaining variables. On the other hand, the interpretation of $CylProjection$ projects the specified variables in the set S and leave the result in the same space. The result of this projection is regarded as "cylindrical". In other words, $Projection_V(S)$ is a subset of $\mathbb{R}^k$ obtained by taking all points $(x_1, \ldots, x_n)$ in S and forgetting all coordinates of variables in V. In contrast, $CylProjection_V(S)$ is the set of points $(x_1, \ldots, x_n)$ in $\mathbb{R}^n$ which can be transformed into points of S by changing values of variables in V. Therefore $CylProjection_V(S)$ is a cylinder in $\mathbb{R}^n$ whose base is $Projection_V(S)$.

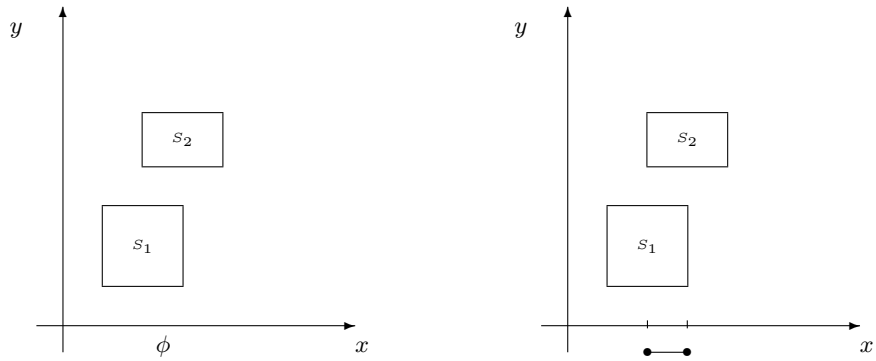
For example, the set $S = \{(x, y, z) : x^2 + y^2 + z^2 - 1 \leq 0\}$ defines a sphere. From the definition above

$$Projection_{\{z\}}(S) = \{(x, y) : x^2 + y^2 - 1 \leq 0\}$$

is a disc in xy-space, whereas

$$CylProjection_{\{z\}} = \{(x, y, z) : x^2 + y^2 - 1 \leq 0\}$$

is a cylinder in xyz-space which has the disc $Projection_{\{z\}}(S)$ as a base.



$Projection_{\{y\}}(S_1 \cap S_2)$ $\qquad$ $Projection_{\{y\}}(S_1) \cap Projection_{\{y\}}(S_2)$

Fig. 4. $Projection_{\{y\}}(S_1 \cap S_2) \neq Projection_{\{y\}}(S_1) \cap Projection_{\{y\}}(S_2)$

5.3 Relationship Between Operators

Subsets of $\mathbb{R}^n$ obtained from extended semi-algebraic primitives using combinations of Boolean operators; n-ary union, n-ary intersection, unary complement, and the two new operators; unary boundary and unary projection, provide finite description of geometric objects and a set of operators capable of manipulating them. The extended CSG system is a Boolean algebra of extended semi-algebraic sets, therefore CSG models is stable since the Boolean operators are supported by well-defined set of axiom. In contrast, we do not at present know how to compute with the extended CSG system with the two new operators. However, we can establish some relationship between the two new operators and the usual Boolean operators as follow:

$$\begin{aligned}
\mathrm{P}rojection_{\{V\}}(S_1 \cup S_2)) &= Projection_{\{V\}}(S_1) \cup Projection_{\{V\}}(S_2) \\
\mathrm{C}ylProjection_{\{V\}}(S_1 \cup S_2)) &= CylProjection_{\{V\}}(S_1) \cup CylProjection_{\{V\}}(S_2) \\
Boundary(S) &= Boundary(complement(S)) \\
Boundary(\phi) &= Boundary(universe) = \phi \\
Boundary(S_1 \cup S_2)) &\subseteq (Boundary(S_1) \cup Boundary(S_2)) \\
Boundary(S_1 \cap S_2) &\subseteq (Boundary(S_1) \cap Boundary(S_2))
\end{aligned}$$

Although the above expressions appear to be true, similar expressions are not. For example, as illustrated in Figure 4.

$$Projection_{\{y\}}(S_1 \cap S_2) \neq Projection_{\{y\}}(S_1) \cap Projection_{\{y\}}(S_2)$$

Similarly,

$$\begin{aligned}
\mathrm{P}rojection_V(S_1 \cap S_2) &\neq Projection(S_1) \cap Projection(S_2) \\
\mathrm{B}oundary(S_1 \cup S_2) &\neq Boundary(S_1) \cup Boundary(S_2) \\
\mathrm{B}oundary(S_1 \cap S_2) &\neq Boundary(S_1) \cap Boundary(S_2)
\end{aligned}$$

6 Results and Observations

The simplest types of Movers' problems are those that concerned with a single moving convex object among known static convex obstacles. We consider several example cases of a single 2-dimensional object avoiding several 2-dimensional static obstacles in a 2-dimensional Workspace. Thus the dimensions of the Omnispaces are between four and six, depending on the degrees of freedom of the object and the resulting C-space. After the projection the resulting C-space is at most 3-dimensional.

The main concern of our experiments is to gain an insight into how to use pruning and spatial subdivision technique by Interval Analysis in conjunction with projection operators on Omnimodels. Algebraically, the projections eliminate two variables from the total of three to five variables.

The experiments were conducted using computer logic system REDLOG implemented in REDUCE computer algebra system. REDUCE is a powerful Computer Algebra system available for many operating systems. The elementary functions are easy to use and for non-interactive mode users can write new procedures using REDUCE syntax. The system was described in more details in, for example [17] [25]. REDLOG which stands for REDuce LOGic system provides an extension to REDUCE computer algebra system so that logical expressions and quantifiers can be dealt with. It provides many functions for the symbolic manipulation of first order formula over some temporarily fixed languages and theories. The focus of the system is on simplification of quantifier-free formula and effective quantifier elimination.

The algorithm was implemented using REDUCE syntax. The procedures are relatively high-level which allow input of the form:

```
procedure_name(object,
               obstacle,
               box,
               number_of_bound_varialble,
               variable);
```

where `object` and `obstacle` are semi-algebraic descriptions, `box` is a list of intervals, `number_of_bound_variable` is an integer and `variable` is a list of variables available to `object` and `obstacle`.

Figure 5 shows the 2-dimensional Workspace which has one moveable object with 2 degrees of freedom. The translation is only allowed in 1-dimension, along the x-axis. The rotation is possible around its reference vertex. The three obstacles are of infinitesimal width. The result Configuration Space obtained from the Omnimodel by using *Projection* operator. The vertical and horizontal lines represent the division decisions that took place. The vertical and horizontal axis corresponds to the rotational range of 2π and the translation range from 0 to 14 respectively. The shaded areas represent C-free while the lighter areas were *undecided*. The recursive subdivision algorithm was adaptive but only terminated at a specified size of boxes. Thus, it was clear from the result that C-space obstacles or the forbidden regions were not detected.

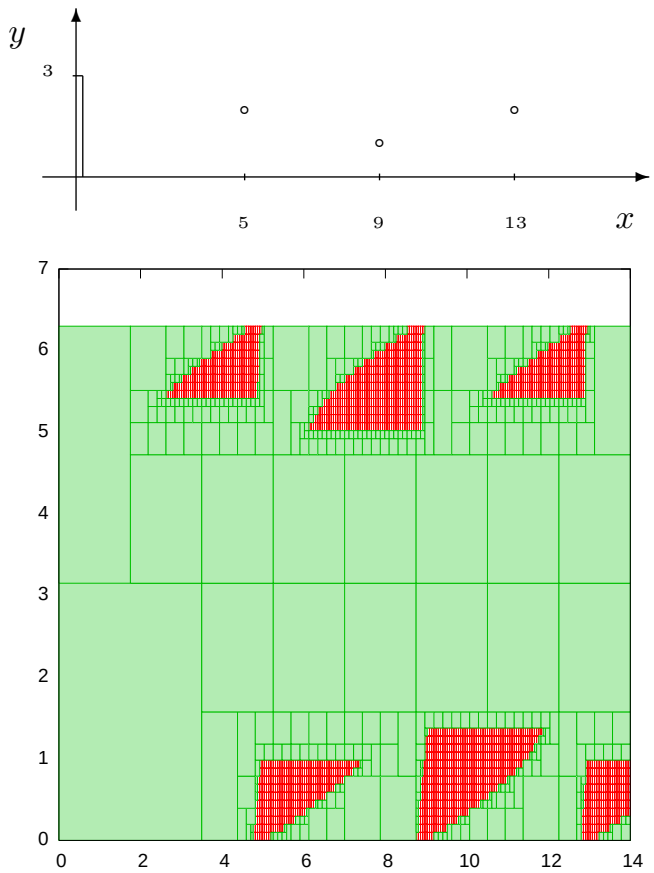


Fig. 5. Workspace and a configuration space of a moving object amidst three point obstacles. The moveable object can translate only in one direction; along the x-axis and can rotate around one end.

This result reveals limitations of standard interval arithmetic similar to those found by Stahl in [45] and similarly by Tupper in [49]. Due to the conservativeness of the interval arithmetic, the evaluation processes could not simplify the Omnimodel to determine if the box is completely inside the C-space obstacles.

6.1 Remarks

When the CSG primitive are extended semi-algebraic sets, pruning and recursive subdivision are still applicable. This is because the evaluation is done by interval arithmetic which *sine* and *cosine* functions are also defined. Thus it seems possible to work directly with the representation of the C-space obstacle as a projection, in an extended CSG system, using Interval Analysis.

Recently, there are many works which address the problem of the overestimation of interval arithmetic. One research direction is to look further into the application of those proposed solutions. In any case, if the moving object or part of the obstacle is not algebraic, C-space obstacle must be represented as a projection, since elimination of quantifiers may not be possible. A useful research topic would be the investigation of an extended Constructive Solid Geometry system which would have both bounded projection and boundary formation as operators, as well as the usual Boolean operators. In such a lazy system, quantifier elimination could be deferred, and might never be carried out.

7 Acknowledgement

This work was conducted during the visit to the Research Institute for Symbolic Computation (RISC), Johannes Kepler University Linz, Austria. The author was supported by the scholarship Technolgiestipendien Südostasien 2005/2006 (Postdoc) awarded by the Austrian Exchange Service – Agency for International Cooperation in Education and Research (ÖAD), Academic Cooperation and Mobility (ACM).

The author would like to express his most sincere gratitude to Professor Dr. Franz Winkler - the host supervisor and the Chairman of the Research Institute for Symbolic Computation (RISC), Professor Dr. Josef Schicho - the Leader of Symbolic Computation Group, Johann Radon Institute for Computational and Applied Mathematics (RICAM), Austrian Academy of Sciences (ÖAW), Professor Dr. Hoon Hong - Department of Mathematics North Carolina State University, Dr. Thomas Sturm, Professor Dr. Volker Weispfenning and Dr. Andreas Dolzmann - Fakultät für Informatik und Mathematik, Universität Passau, Germany and Mr. Hirokazu Anai - Fujitsu Laboratories Ltd., Japan.

References

1. N. Amato and L Dale. Probabilistic Roadmap Methods are Embarrassingly Parallel. Technical Report TR98-024, Department of Computer Science, Texas A&M University, 25, 1998.
2. P. Berman, B. DasGupta, and S. Muthukrishnan. Exact size of binary space partitionings and improved rectangle tiling algorithms. *SIAM J. Discret. Math.*, 15(2):252–267, 2002.
3. A. Bowyer, J. Berchtold, D. Eisenthal, I. Voiculescu, and K. Wise. Interval methods in geometric modeling. In *GMP '00: Proceedings of the Geometric Modeling and Processing 2000*, page 321, Washington, DC, USA, 2000. IEEE Computer Society.
4. M. Branicky, S. Lavalle, K. Olson, and L. Yang. Quasirandomized path planning, 2001.
5. M. Branicky, S. LaValle, K. Olson, and L. Yang. Deterministic vs. probabilistic roadmaps, 2002.
6. M. S. Branicky and W. S. Newman. Rapid Computation of Configuration Space Obstacles. *Proc 1990 IEEE International Conference Robotics Automation*, pages 304–310, 1990.
7. R. A. Brooks and T. Lozano-Pérez. A Subdivision Algorithm in Configuration Space for Findpath with Rotation. *IEEE Transactions on Systems Man and Cybernetics*, 15(2):224–233, 1985.
8. J. F. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, Mass., 1988.
9. G. E. Collins. Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition. In *Proceedings of the 2nd GI Conference*, volume 33 of *Lecture Notes in Computer Science*, pages 134–183. Springer, Berlin, 1975.
10. J. H. Davenport. On a 'Piano Movers' Problem. *ACM SIGSAM Bulletin*, pages 15–17, 1986.
11. J. H. Davenport, Y. Siret, and E. Tournier. *Computer Algebra: Systems and Algorithms for Algebraic Computation*. Academic Press, 1988.
12. T. Duff. Interval arithmetic recursive subdivision for implicit functions and constructive solid geometry. In *SIGGRAPH*, pages 131–138, 1992.
13. A. Dumitrescu, J. S. G. Mitchell, and M. Sharir. Binary space partitions for axis-parallel segments, rectangles, and hyperrectangles. In *SCG '01: Proceedings of the seventeenth annual symposium on Computational geometry*, pages 141–150, New York, NY, USA, 2001. ACM Press.
14. B. Faverjon. Obstacle Avoidance Using an Octree in the Configuration Space of a Manipulator. In *Proceedings of the IEEE Int. Conf. Robotics, Atlanta, GA*, pages 504–512, March 1984.
15. H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by a priori tree structures. In *SIGGRAPH '80: Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, pages 124–133, New York, NY, USA, 1980. ACM Press.
16. A. J. Gomes and J. C. Teixeira. A mathematical framework for set-theoretic solid models. In *CSG 94 Set-theoretic Solid Modelling: Techniques and Applications*, pages 19–33. Information Geometers, 1994.
17. A. C. Hearn. *REDUCE User's Manual Version 3.7*. RAND Publication CP78, 1999.
18. H. Hong. *Improvements in CAD-based Quantifier Elimination*. PhD thesis, Ohio State University, 1990.

19. H. Hong. Approximate Quantifier Elimination. In *Proceedings of SCAN'95*, 1995. Invited Talk.
20. J. E. Hopcroft, J. T. Schwartz, and M. Sharir. On the Complexity of Motion Planning for Multiple Independent Objects: P-space-hardness of the "Warehouseman's Problem". *International Journal of Robotics Research*, 3(4):76, 1984.
21. Y. K. Hwang and N. Ahuja. Gross Motion Planning — a Survey. *ACM Computing Surveys*, 24(3):219–291, 1992.
22. M. Kalkbrener and S. Stifter. Some examples for using quantifier elimination for the collision problem. Technical Report 87-22, Research Institute for Symbolic Computation, J. Kepler University, 1987.
23. L. Kavraki, P. Svestka, J. Latombe, and M. Overmar. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. Technical Report CS-TR-94-1519, Department of Computer Science, Rice University, 1994.
24. T. Lozano-Pérez. Spatial planning: a configuration space approach. *IEEE Transactions on Computers*, 32(2):108–119, 1983.
25. M. A. H. MacCallum and F. J. Wright. *Algebraic Computing with REDUCE*. Oxford University Press, 1991.
26. T. Martin, H. Shou, I. Voiculescu, A. Bowyer, and G. Wang. Comparison of interval methods for plotting algebraic curves. *Comput. Aided Geom. Des.*, 19(7):553–587, 2002.
27. S. McCallum. Partial Solution to a Path Finding Problem Using the CAD Method. Technical report, Department of Computing, Macquarie University, Australia, June 1995.
28. R. E. Moore. *Interval Analysis*. Prentice-Hall, Inc., 1966.
29. S. Parry-Barwick and A. Bowyer. Multidimensional set-theoretic feature recognition. *Computer-Aided Design*, 27(10):731–740, 1995.
30. S. J. Parry-Barwick. *Multi-dimensional Set-theoretic Geometric Modelling*. PhD thesis, Department of Mechanical Engineering, University of Bath, 1995.
31. S. J. Parry-Barwick and A. Bowyer. Woodwark's Method for Feature Recognition. *Technical Report 099/1992*, 1992.
32. M. S. Paterson and F. Frances Yao. Efficient binary space partitions for hidden-surface removal and solid modeling. *Discrete Comput. Geom.*, 5(5):485–503, 1990.
33. M. S. Paterson and F. Frances Yao. Optimal binary space partitions for orthogonal objects. In *SODA '90: Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 100–106, Philadelphia, PA, USA, 1990. Society for Industrial and Applied Mathematics.
34. S. Ratschan. Approximate Qunatified Constraint Solving By Cylindrical Box Decomposition. Technical Report 00-27, Research Institute for Symbolic Computation (RISC), Linz., September 2000.
35. S. Ratschan. Convergence of Approximate Constraint Solving by Approximate Quantifiers. Technical Report 00-23, Research Institute for Symbolic Computation (RISC), Linz., June 2000.
36. S. Ratschan. Uncertainty Propagation in Heterogenous Algebras for Approximate Quantified Constraint Solving. Technical Report 00-23, Research Institute for Symbolic Computation (RISC), Linz., September 2000.
37. J. H. Reif. Complexity of the Mover's Problem and Generalizations. *Proceedings of the 20th Symposium on the Foundations of Computer Science*, pages 421–427, 1979.
38. J. H. Reif and M. Sharir. Motion Planning in the Presence of Moving Obstacles. *Journal Of The Association For Computing Machinery*, 41(4):764–790, 1994.

39. J. T. Schwartz and M. Sharir. On the piano movers' problem I: The case of a two-dimensional rigid polygo nal body moving amidst polygonal barriers. *Comm on Pure & Applied Maths*, 36:345, 1983.
40. J. T. Schwartz and M. Sharir. On the "piano movers" problem II. general techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, 4:298–351, 1983.
41. J. T. Schwartz and M. Sharir. On the piano movers' problem III: Coordinating the motion of several independent bodies: the special case of circular bodies moving amidst polygonal barriers. *International Journal of Robotics Research*, 2(3):46, 1983.
42. J. T. Schwartz and M. Sharir. On the piano movers' problem V: The case of a rod moving in three-dimensional space amidst polyhedral obstacles. *Comm of Pure & Applied Maths*, 37:815, 1984.
43. M. Sharir and E. A. Sheffi. On the piano movers' problem IV: Various decomposable-dimensional motion-planning problems. *Comm on Pure & Applied Maths*, 37:479, 1984.
44. J. M. Snyder. Interval analysis for computer graphics. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 121–130, New York, NY, USA, 1992. ACM Press.
45. V. Stahl. *Interval Methods for Bounding the Range of Polynomials and Solving Systems of Nonlinear Equations.* PhD thesis, RISC, Johannes Kepler University Linz, Austria, 1996.
46. T. Sturm and V. Weispfenning. Computational Geometry Problems in REDLOG. Technical Report MIP-9708, Fakultät für Mathematik und Informatik, Universitä t Passau, April 1997.
47. A. Tarski. *A Decision Method for Elementary Algebra and Geometry.* University of California, Berkeley, 1951.
48. W. C. Thibault and B. F. Naylor. Set operations on polyhedra using binary space partitioning trees. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 153–162, New York, NY, USA, 1987. ACM Press.
49. J. Tupper. Reliable two-dimensional graphing methods for mathematical formulae with two free variables. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 77–86, New York, NY, USA, 2001. ACM Press.
50. S. M. Udupa. *Collision detection and avoidance in computer controlled manipulators.* PhD thesis, Department of Electrical Engineering, California Institute of Technology, 1977.
51. A. F. Vanderstappen, D. Halperin, and M. H. Overmars. The Complexity of the Free-space for a Robot Moving Amidst Fat Obstacles. *Computational Geometry — Theory and Applications*, 3(6):353–373, 1993.
52. K. D. Wise. Using multidimensional CSG models to map where objects can and cannot go. Technical Report 001/96, Department of Mechanical Engineering, University of Bath, January 1996.
53. K. D. Wise. *Computing Global Configuration-Space Maps Using Multidimensional Set-Theoretic Modelling.* PhD thesis, University of Bath, 1999.